Modelación de sistemas multiagentes con gráficas computacionales

Gpo 570

Evidencia 2

Movilidad Urbana

Alejandra Coeto Sánchez - A01285221

Alonso Huerta Escalante - A00836072

Luis Adrián Amado Álvarez A01571393

Monterrey, NL.
5 de febrero, 2025

Índice

1. Introducción

La simulación de multiagentes (MAS por sus siglas en inglés) es un enfoque computacional donde se crean agentes individuales con comportamientos autónomos que interactúan en un entorno simulado. Los agentes pueden representar: Personas, vehículos, semáforos y cualquier entidad relevante.

La movilidad urbana enfrenta diversos retos, y las MAS ofrecen soluciones adaptativas y dinámicas en áreas clave, como la congestión del tráfico, planificación de rutas, eficiencia del transporte público, seguridad peatonal y el flujo peatonal.

Un modelo virtual dinámico que imita las interacciones de los elementos del sistema ofrece una plataforma para probar políticas, tecnologías y diseños urbanos antes de implementarlos en el mundo real. Entre otras cosas, permite la representación realista del entorno urbano, análisis de interacciones complejas, experimentación sin riesgos reales, optimización de recursos urbanos y planificación de escenarios futuros.

2. Créditos

Nombre	**Puesto**
Alonso Huerta Escalante	Programación
Luis Adrián Amado Álvarez	Programación
Alejandra Coeto Sánchez	Programación

Nombre	**Fortalezas**	**Área de oportunidad**	**Expectativas**
Alonso Huerta	Tengo un poco de experiencia haciendo multithreading en python, lo que puede ayudar con la eficiencia	Nunca he interactuado con modelación 3D, mucho menos con simuladores	Al final de este bloque espero poder ser competente en la simulación de agentes inteligentes, a tal punto que pueda hacer simulaciones a gran escala. Espero poder tener un conocimiento competente de modelación 3D.
Alejandra Coeto	Experiencia en trabajos colaborativos y el uso de herramientas con python.	Poca experiencia con Unity y no tengo experiencia en simulaciones en tres dimensiones	Me gustaría entender los procesos multiagentes a nivel conceptual y de implementación. De igual manera me interesaría conocer más de deep learning y poder

			implementarlo en la solución del reto.
Luis Amado	Tengo experiencia con diferentes herramientas de python.	No tengo experiencia trabajando con el proceso de modelado en 3 dimensiones. Tampoco he trabajado mucho con Unity.	De este bloque espero aprender a utilizar las simulaciones con agentes de manera eficiente para poder crear diferentes ambientes y probar combinaciones de parámetros que me puedan ayudar a hacer decisiones en la vida real.

3. Contexto y problema

Uno de los problemas más grandes de la movilidad urbana en términos de transporte privado mediante automóviles, es la necesidad de que existan intersecciones entre distintos flujos de tráfico. Hacer esto de una manera inteligente, no solo ayuda a reducir el tiempo de viaje para los pasajeros, sino que también nos ayuda a reducir aspectos como la contaminación y mejorar la seguridad de los peatones.

4. Objetivos

El objetivo de la investigación es diseñar y evaluar un modelo de movilidad urbana basado en sistemas multiagente para maximizar el flujo vehicular en una ciudad pequeña, analizando el impacto de diferentes estrategias de gestión de tráfico en la eficiencia del transporte diario.

El software simulará un escenario con casas y destinos, donde agentes peatones deben encontrar un camino desde su origen a su destino y caminar para llegar a él de manera eficiente y siguiendo las reglas de la calle como la señalización de los semáforos. El objetivo específico de este proyecto es realizar esta simulación de manera que podamos monitorear cambios en ciertos indicadores al modificar algunos parámetros.

5. Indicadores y parámetros

Los parámetros nos permiten cambiar las condiciones de la simulación y los indicadores nos ayudan a materializar y cuantificar los impactos que tienen los cambios de nuestros parámetros en los trayectos de los peatones.

Indicadores

- Tiempo promedio de viaje: Tiempo que toma a un vehículo desplazarse de su origen a su destino.

- Velocidad promedio: Velocidad efectiva de los peatones (medida de la distancia desplazada y el tiempo tardado en llegar).
- Tiempo de espera promedio en intersecciones: Cantidad de pasos que pasa un peatón esperando en un cruce peatonal.

Parámetros
Algunos de los parámetros que dejamos abiertos para modificaciones son la densidad de vehículos, la posición y cantidad de semáforos, la estructura general del ambiente y ciertos parámetros de aprendizaje como los valores alpha, gamma y epsilon.

Sin embargo, el parámetro que consideramos más importante específicamente para nuestra simulación de peatones es la duración (en pasos) de la luz verde en los semáforos.

6. Historias de usuario

ID	Historia de Usuario
HU1	Como cliente, quiero visualizar simulaciones de movilidad urbana para maximizar el flujo vehicular y diseñar estrategias para gestionar el tráfico.
HU2	Como cliente, quiero que se simule la interacción de peatones, vehículos y semáforos para analizar un entorno de movilidad urbana.
HU3	Como cliente, quiero que los peatones se muevan únicamente por las banquetas, zonas de parque e intersecciones, para simular sus movimientos regulares.
HU4	Como cliente, quiero que los vehículos se muevan únicamente por las calles y se detengan en las intersecciones o peatones para simular el tráfico.
HU5	Como cliente, quiero que los semáforos se encuentren programados para gestionar la intersección principal, dando paso a los vehículos y peatones de manera estructurada.

7. Descripción del sistema multiagente

7.1. Agentes

Peatones
Los agentes de peatones son agentes inteligentes. Son agentes basados en el objetivo ya que tienen un destino específico al que quieren llegar.

<u>Acciones de los peatones:</u>
- Caminar hacia arriba
- Caminar hacia abajo

- Caminar hacia la izquierda
- Caminar hacia la derecha
- Esperar un paso

Estados de los peatones:
- Posición
- Destino elegido

Variables de los peatones:
- Modo entrenamiento
- Modo intersección
- Tiempo de espera
- Distancia recorrida

Vehículos

Los agentes de vehículos son igualmente agentes inteligentes puramente reactivos. Su dirección se escoge de manera aleatoria y siguen las leyes de la calle.

Acciones de los vehículos:
- Moverse en la dirección de la calle
- Esperar un paso
- Cruzar una intersección hacia adelante
- Cruzar una intersección hacia su izquierda
- Cruzar una intersección hacia su derecha

Estados de los vehículos:
- Posición
- Dirección a la que quiere girar

Variables de los vehículos:
- Lista de movimientos para cruzar la intersección (está vacía cuando no está en una intersección)

Semáforos

Agentes triviales (no inteligentes) con dos posibles estados y una posición, los cuales cambian acorde al tiempo.

Acciones de los semáforos:
- Activar un cruce peatonal
- Desactivar el cruce peatonal
- Mover al siguiente espacio
- Activar cooldown
- Esperar un paso

Estados de los semáforos:

- Posición
- Tiempo de cooldown

Variables de los semáforos:

- Activar un cruce peatonal
- Desactivar el cruce peatonal
- Mover al siguiente espacio
- Activar cooldown
- Esperar un paso

7.2. Ambiente

El ambiente es una cuadrícula (Grid) con estados. Tendrá obstáculos, cruces, calles, edificios y banquetas. Los cruces tienen un estado adicional que determina si está activado o desactivado para los peatones. El semáforo tiene la responsabilidad de activar y desactivar los cruces. Todos los agentes tienen acceso completo a la información de la cuadrícula con la que pueden planear su ruta. Por lo tanto, nuestro ambiente es discreto, dinámico y accesible.

Para más información, vea la Sección 8.1.

7.3. Interacciones entre agentes y ambiente

Cada tipo de agente tiene tipos de celdas del ambiente dentro de las cuales tienen permitido transitar. Los peatones pueden pasar sobre las banquetas y los cruces peatonales. Los vehículos pueden pasar sobre las calles y los cruces peatonales. Además, si un vehículo está sobre una celda de tipo calle, ese vehículo solo se podrá mover en la dirección que lo indica esa celda.

Los comportamientos de los agentes móviles (peatones y vehículos) cambia dependiendo de la posición de otros agentes:

- Peatones

Los peatones no pueden pasar por una casilla que tiene un carro o un cruce peatonal desactivado. Sin embargo, si un agente desea pasar por una casilla que contiene otro peatón, lo puede hacer sin problema.

- Vehículos

Los vehículos no pueden cruzar un cruce peatonal si hay peatones que lo están cruzando. Está prohibido que dos vehículos ocupen la misma celda y por lo tanto si el vehículo quiere pasar por una celda que contiene otro vehículo, no lo podrá hacer en ese paso y se tendrá que

esperar. Además, no puede entrar a la intersección si el semáforo se encuentra actualmente en rojo o amarillo (periodo de cooldown).

7.4. Entrenamiento y planeación de rutas

Los agentes de peatones encuentran la ruta de su origen a su destino mediante un entrenamiento con el algoritmo de Q-learning. Este entrenamiento se realiza en un ambiente estático inicial, donde todos los cruces están activados y se ignoran las posiciones de los vehículos y los estados de los semáforos. Como fue mencionado anteriormente, el estado de los peatones se compone no solo de su posición sino también del destino al que se busca llegar. Esto nos permite tener una sola tabla-Q que todos los peatones pueden compartir. Cómo el estado incluye el destino del agente, es como si cada grupo de peatones con el mismo destino tuviera su propia tabla-Q y por lo tanto están trabajando juntos y comunicándose de manera indirecta para encontrar sus rutas respectivas al destino en común. Esta tabla-Q se guarda dentro de una clase, la cual todos los peatones actualizan y consultan durante su entrenamiento, por lo tanto se puede considerar como un Blackboard de información compartida.

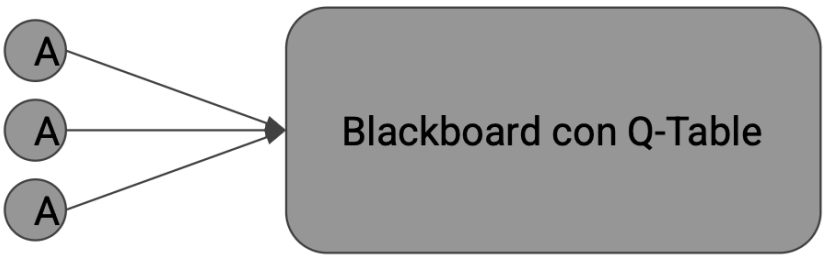


Imagen 1: Visualización de agentes y Blackboard

A continuación se puede observar las diferentes puntuaciones y castigos que se le asignan a los agentes dependiendo del tipo de celda.

Tipo de celda	**Puntuación/Castigo**
Edificios y obstáculos	-1000
Calles	-500
Pasto	-20
Banquetas y cruces	-1
Destino	1000

7.5. Diagramas de los agentes

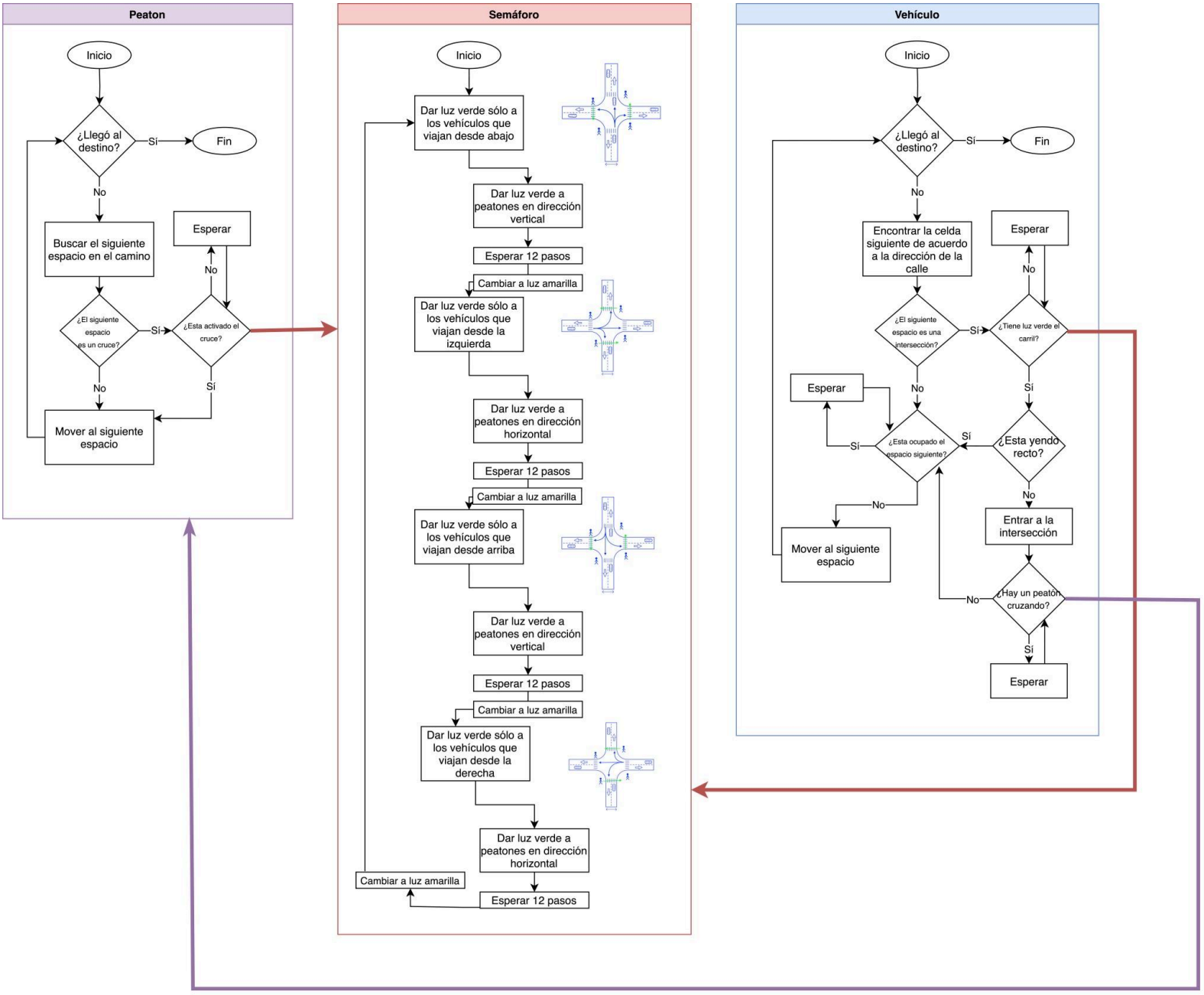


Imagen 2: Diagrama de interacciones entre agentes

Versión grande

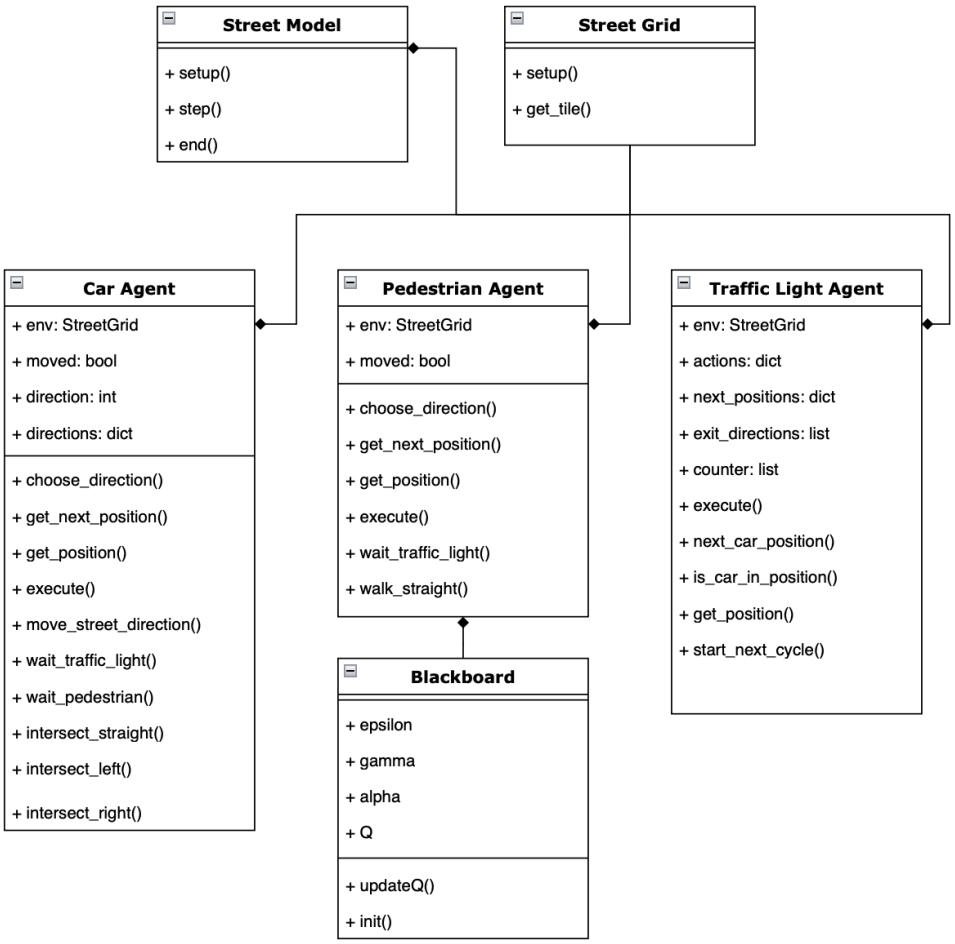


Imagen 3: Diagrama de clases

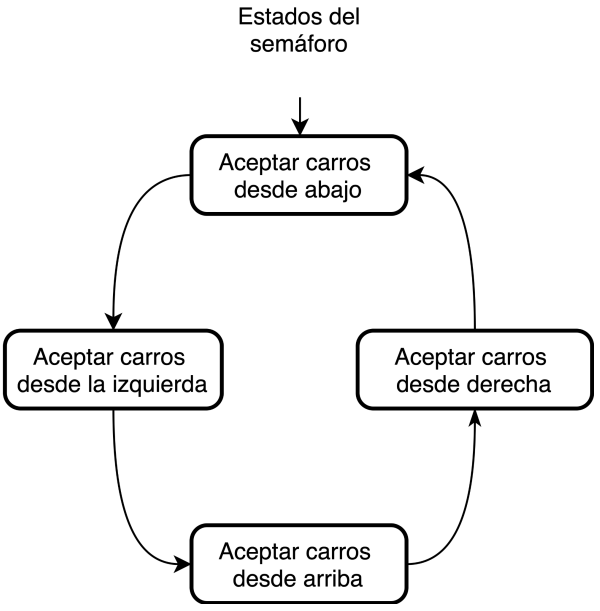


Imagen 4: Diagrama de estados del semáforo

8. Descripción del modelo gráfico

8.1. Escena a modelar

Propuesta de escena

Una **cuadrícula** con distintos estados

- *Casas (Rojo):* una cuadrícula donde habitan los agentes
- *Destinos (Azul):* lugar a donde quieren llegar los agentes
- *Calles (Gris oscuro con flechas):* lugares donde se pueden mover los agentes de vehículo
- *Banquetas (Azul grisáceo):* lugares donde se pueden mover los agentes de peatones
- *Cruces peatonales (Activado: Gris claro, Desactivado: Blanco):* lugares donde se pueden mover los agentes de vehículo y los agentes de peatones
- *Obstáculos (Estacionamiento, obstáculo, pasto):* lugares que no clasifican como ninguno de los anteriores

Propuesta de ambiente

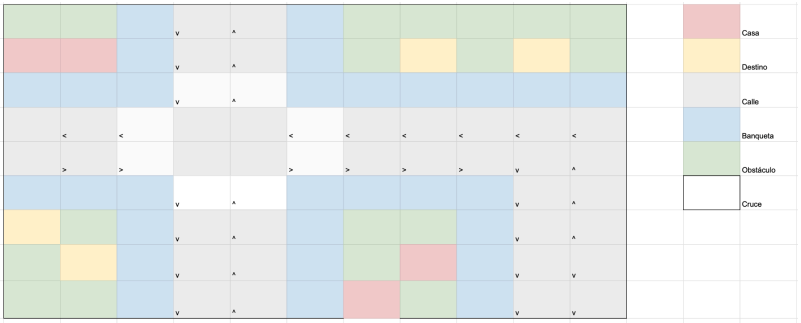


Imagen 5: Primera propuesta del ambiente en 2 dimensiones.

Propuesta preliminar con AgentPy

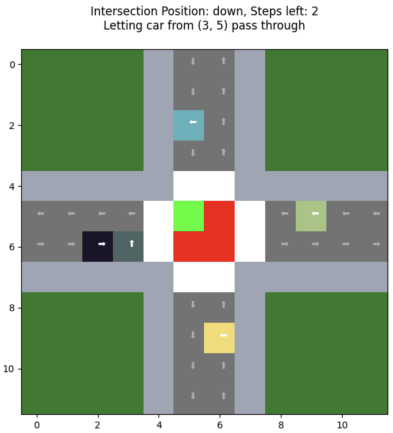


Imagen 6: Primera propuesta de la escena en 2 dimensiones.

Dentro de esta primera iteración, se incluye una sola intersección, controlada por un semáforo. Este ambiente es de tipo Torus de manera que los carros siguen circulando. Incluimos únicamente a los vehículos para esta demostración pero ya tenemos colocados los

cruces peatonales, como se muestra en blanco. Cada uno de los 5 vehículos aparece y decide qué salida de la intersección va a tomar. Antes de chocar con un carro o entrar a la intersección, primero revisa que lo pueda hacer de manera segura (que no haya un carro en frente o que el semáforo esté en verde).

Diseño final con AgentPy

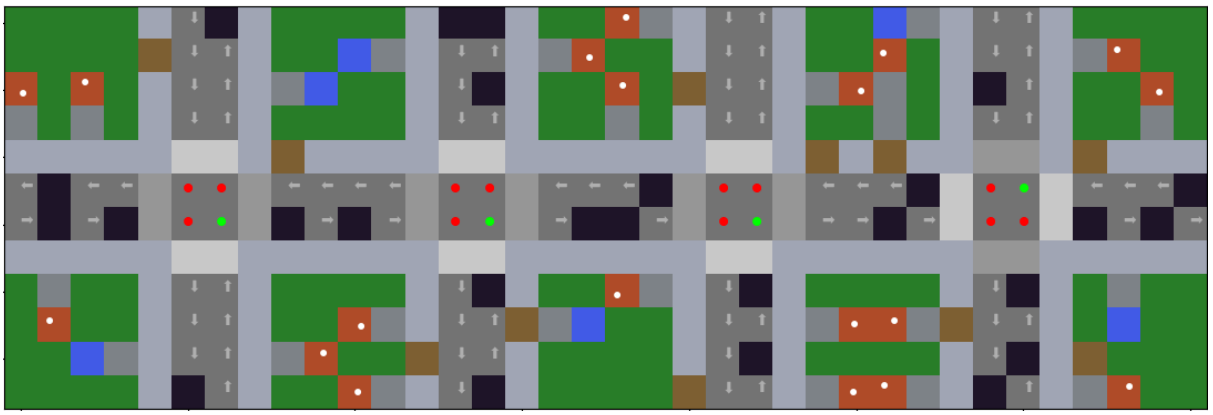


Imagen 7: Diseño final con AgentPy.

Para la simulación final, se le agregaron los agentes de los peatones, con sus hogares y destinos. También se modificó la visualización para que se vean más fácilmente los diferentes tipos de agentes. En este caso, los peatones se ven como círculos blancos y están posicionados de manera que dos agentes se pueden distinguir aún cuando se encuentran en la misma celda.

Video de la simulación final: https://youtu.be/FvU5_ryqb8M

Propuesta preliminar en Unity

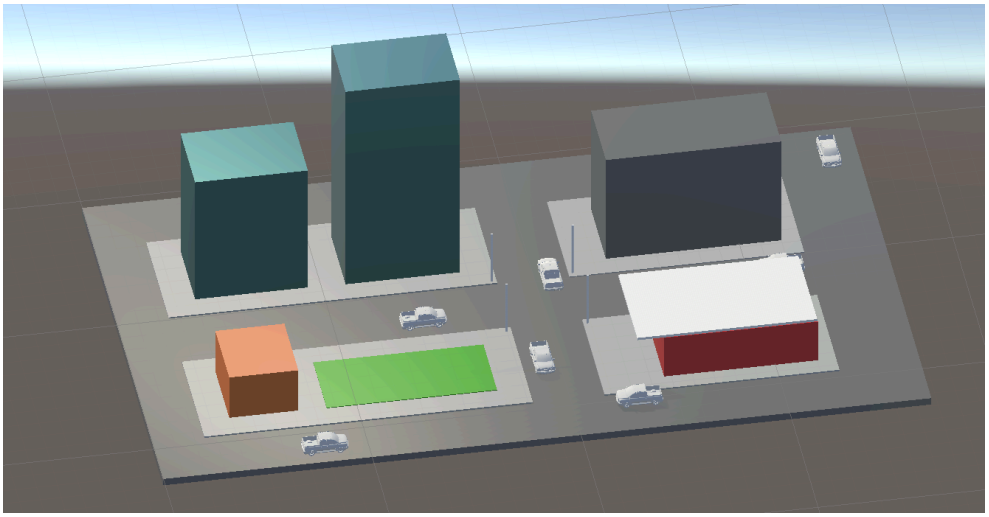


Imagen 8: Diseño modelado en Unity.

Para la primera propuesta se realizó el plano principal con algunas banquetas y bloques que representan edificios. De igual manera se seleccionó el modelo que se utilizará para los vehículos. No obstante es un diseño considerablemente simple con un propósito de exploración dados los cambios esperados en el mapa.

Diseño final en Unity

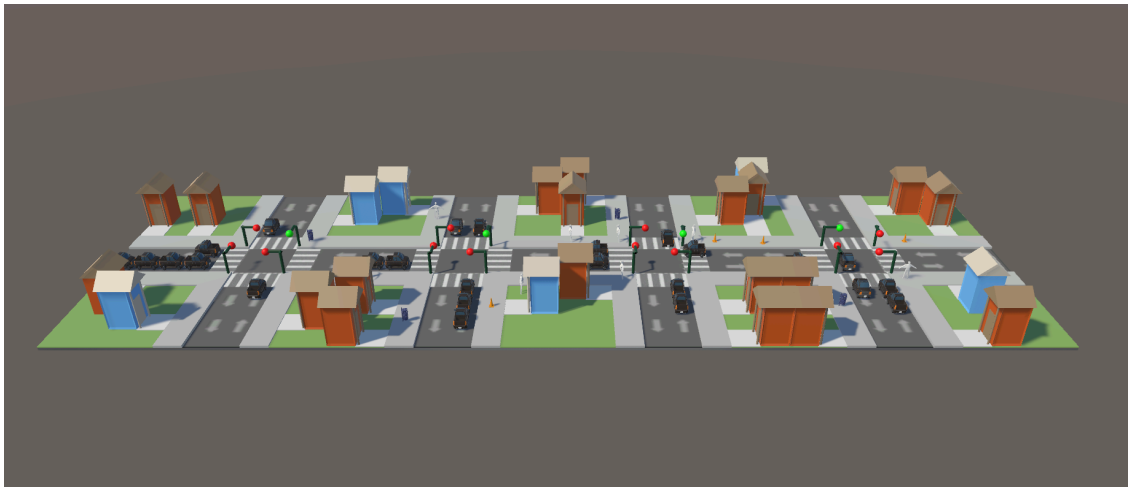


Imagen 9: Diseño final en Unity con texturas.

Para el diseño final, se realizó el plano principal y se le asignó una textura generada en Canva para mostrar las flechas y los cruces peatonales (mostrada a continuación). De igual manera se agregaron las banquetas, zonas verdes y caminos usando las figuras básicas de Unity. Finalmente, para el diseño de las casas y semáforos se utilizó la herramienta Pro-Builder.

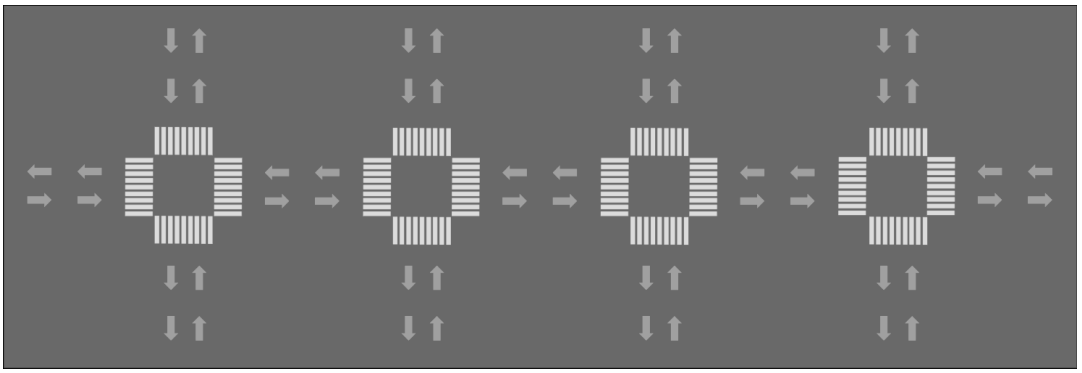


Imagen 10: Imagen usada como textura para el plano principal.

Por otro lado, para los obstáculos, vehículos y peatones se utilizaron modelos en formato obj como se indica en la sección 8.2 y 8.3. Adicionalmente, se crearon prefabs para renderizar a los agentes mediante un script y así instanciar o actualizar sus posiciones, orientaciones y color en el caso de los semáforos.

En cuanto a la lógica de los agentes, cada uno tiene un método que recibe la información de la API y verifica si el agente existe para determinar si instanciarlo o actualizar su información usando sus vectores de posición y rotación.

Comunicación AgentPy - Unity

Para la comunicación entre la interfaz gráfica y la lógica, se implementó una API usando Flask, la cual cuenta con varios endpoints para mandar información de la simulación. Por lo tanto, en Unity mediante una corutina se llama a la API para recibir la información de los agentes en cada paso y se actualiza gráficamente. A continuación se muestran los endpoints principales:

/mobile_agent_step/{step_number}[GET]

Método para obtener las localizaciones de los agentes.

```javascript
[
  {
    'id': int,
    'x': int,
    'y': int,
    'type': string
  },
  ...
]
```

/static_agent_step/{step_number}[GET]

Método para obtener las localizaciones de los semáforos y su estado.

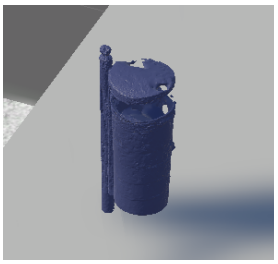
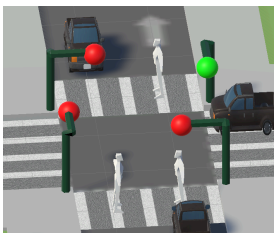
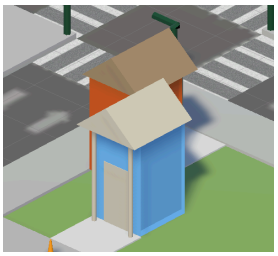
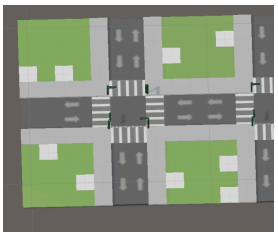
```javascript
[
  {
    'id': int,
    'x': int,
    'y': int,
    'status': bool (True = Green, False = Yellow)
  },
  ...
]
```

/simulation_stats [GET]

Método para obtener estadísticas meta de la simulación.

```javascript
{
    'traffic_light_step_duration': int,
    'traffic_light_step_cooldown': int,
    'car_density': double,
    'steps': int,
    'alpha': double,
    'gamma': double,
    'epsilon': double,
    'train_episodes': 200,
    'car_id_list': [int],
    'pedestrian_id_list': [int],
    'traffic_light_id_list': [int],
}
```

8.2. Componentes gráficos

Nombre	Descripción	Imagen	Fuente
Conos	Obstáculo		https://www.fab.com/listings/b0fc635c-325c-424b-be27-a9efb0911bf8
Basurero	Obstáculo		https://www.fab.com/listings/525d1af2-2976-446d-9157-786da52501d3
Semáforo (base)	Cilindro que sostiene las luces de los semáforos		Diseñado por el equipo
Casa	Origen o destino para algunos agentes		Diseñado por el equipo
Plano	Piso con flechas, cruce peatonal, banquetas y pasto		Diseñado por el equipo

8.3. Prefabs

Nombre	Descripción	Imagen	Fuente	Scripts
Carro	Agente que se puede mover en las calles		https://fab.com/s/a1d1ad79cffe	AgentController
Peatón	Agente que se puede mover libremente en el plano		https://www.fab.com/listings/8145184b-1bd0-40e8-a9c7-2dd3cf0ab4f5	AgentController
Semáforo	Luz del semáforo. Agente que cambia de color para indicar el paso de los agentes móviles		Diseñado por el equipo	StaticAgentController

8.4. Scripts

Nombre	Descripción	Interacciones
APICaller	Script para realizar llamadas a la API de simulación con Agentpy mediante una corrutina.	Contiene referencias a los scripts de los agentes para que en cada paso se actualicen sus estados.
AgentController	Script con funciones para instanciar agentes nuevos o actualizar sus posiciones y orientaciones.	Contiene referencias a los prefabs de peatón y carro. En caso de no existir previamente, se crean los elementos o en caso contrario se actualizan sus estados.
StaticAgentController	Script con funciones para instanciar los semáforos y actualizar su color.	Contiene referencia al prefab esférico para que se actualice el color.
MobileAgentData	Clase para representar la	NA

	información de un agente móvil.	
StaticAgentData	Clase para representar la información de un agente estático (semaforo)	NA

8.5. Demos

Vista regular con terminal (API): https://youtu.be/ttduSHiK1g0
Vista superior: https://youtu.be/Qz2PEJvJS6g
Vista frontal: https://youtu.be/i_RkS7PlOcI

9. Resultados y conclusiones

Con el código final, se corrieron dos simulaciones, variando el tiempo de duración del semáforo en verde.

Duración de semáforo	12 pasos (+4 cooldown)	4 pasos (+4 cooldown)
Tiempo promedio de viaje	36.32 pasos	31.32 pasos
Velocidad promedio	0.67 celdas por paso	0.79 celdas por paso
Tiempo de espera promedio	15.84 pasos	9.16 pasos

Tomando en cuenta los resultados obtenidos, se puede observar que como era de esperar, los agentes no se mueven en promedio a una velocidad de una celda dado que en algunas ocasiones deben esperar al semáforo. De igual manera, es importante mencionar que el entrenamiento de los peatones no considera los semáforos, por lo que la ruta óptima seleccionada se basa únicamente en la distancia. Esto puede significar que haya rutas más rápidas si se considera el tiempo de espera de los semáforos. Finalmente, como puede observarse en la tabla de resultados, se puede concluir que cuando la duración del semáforo en verde es menor, los agentes esperan en promedio menos tiempo, por lo que llegan más rápido a sus destinos.

10. Entregables de administración del proyecto

	Title	Assignees	Status	Start date	End date
1	M5. Avance 3	Ale-Coeto, alonso284, and luis-amado	Done	Jan 28, 2025	Feb 5, 2025
2	Generar y recopilar assets para Unity	Ale-Coeto	Done	Jan 28, 2025	Feb 3, 2025
3	Agregar peatones a la simulación de AgentPy	luis-amado	Done	Jan 28, 2025	Jan 30, 2025
4	Crear el código para sincronizar Unity con el API	Ale-Coeto and alonso284	Done	Jan 30, 2025	Feb 4, 2025
5	Crear el código del API en Flask	alonso284	Done	Jan 29, 2025	Jan 31, 2025
6	Recopilar información de los indicadores de la simulación	luis-amado	Done	Feb 4, 2025	Feb 5, 2025
7	Funcionalidad de los cruces peatonales	alonso284	Done	Jan 28, 2025	Jan 30, 2025
8	Entrenamiento y aprendizaje de ruta para los peatones	luis-amado	Done	Jan 29, 2025	Jan 30, 2025
9	Actualizar ambiente de Unity	Ale-Coeto	Done	Jan 28, 2025	Feb 5, 2025
10	M5. Presentación	Ale-Coeto, alonso284, and luis-amado	Done	Feb 3, 2025	Feb 5, 2025

Imagen 11: Vista de las tareas de la última entrega dentro del Github Project

Liga de Github Project

Códigos:
Link del Github: https://github.com/alonso284/UrbanMobilityMultiAgents
Link de la simulación preliminar en AgentPy:
https://colab.research.google.com/drive/1YfGqZ41HiJZx13s1l8Qke2vbWJmXQMqS?usp=sharing

11. Referencias

Fab (2024). Discover. Recuperado de: https://www.fab.com/
Foramitti, J., (2021). AgentPy: A package for agent-based modeling in Python.
Journal of Open Source Software, 6(62), 3065, https://doi.org/10.21105/joss.03065